Technical Evaluation

Dear almost-Qtealer,

If you read this, it means we already like you and how you interact with others. In other words, it means we are convinced that you will fit nicely into the Qteal team. However, you are not there yet… We also need to know whether your technical skills are on point. Which is what we are about to "test" right now.

The concept

Together with one of us, you will go through a series of technical challenges. These challenges are small exercises to check how much you know about certain topics. No need to panic at all, we look at more than only the solution. Your way of working, your reasoning and how you get the necessary information to solve the challenges are also taken into count. It is important to know that one might not know everything, so we are there to help/guide you where necessary.

Preparation of the development environment

Before starting the challenges, the following development environment needs to be set up:

- Docker installed on a Linux host (preferred)
    - Or alternatively Docker Desktop can be installed on a Mac or Windows host
- Git is installed on the host

The rules

1. There are no rules[1].
2. You get **90 minutes** to complete as many challenges as possible.
3. You are allowed to **use whatever source you like** to solve the problems that you face.
4. You **do not speak of the challenge** to other people (so that it remains challenging to other people as well).

Before you start, a word of comfort

Try to keep calm all the time. And above all else: "Google is your friend". Good luck!

[1] Except rule number 2, 3 and 4

Challenges

Exercise 1 - Docker image

For this exercise we will set up the necessary docker image for you to use during the remainder of the challenges. If you have never heard of docker: no problem at all! Docker is a set of software packages that provides us with lightweight virtual machines. You might have heard of Virtualbox already? Well, Docker is something similar, but then way cooler. In order to set up the necessary items, run the following commands from a terminal:

```
$ ./start_docker.sh
```

Exercise 2 - Some bash basics

If you take a look into the repository that you just cloned, you can see that there are two different folders: "c_programming" and "python_progamming". If you look into the python folder, you will see a massive amount of python files. You will need one of those. Which one it is, is for you to find out. Some of the files contain complete rubbish data, only a few of them will contain useful data. You can find out which file you need. If you look at the naming of the files, you see that all of them end with a number and ".py" extension. To find the correct number, count the number of occurrences of the word "syzygy" in all of the files of the entire repository that you just cloned. The number of occurrences will determine which file you need for challenge number 3.

Exercise 3 - Python command line arguments

The number you found in the previous exercise is the number of the python file we need for this one. E.g. 'fileX.py' . If you open that python file, you will see at the bottom of the file a 'TODO' to add some command line argument parsing. The file contains 2 functions: a main function and another function. In the main function, the other function should be called with an argument. The argument passed on to the function is "9f64b37fe088cea42a3c4a868a278b2d7dd76f2a" for the first time and "9f64bd7db4dbd8b3313447cac624832d73d066" for the second time. Whatever the code returns after running, you will need for a later exercise.
So summarized: uncomment the code in the bottom of the file, and add the necessary missing code in order for you to be able to run the program as follows:

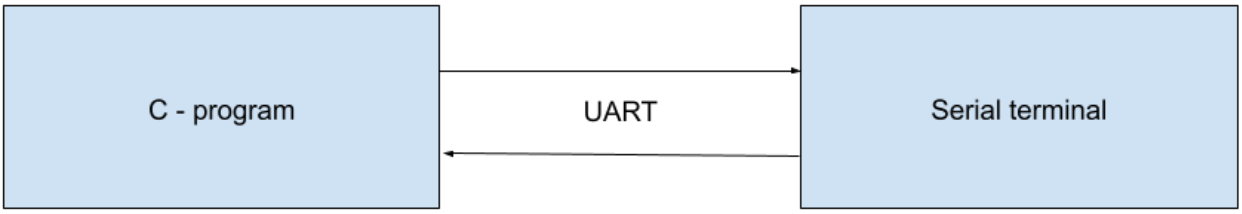
python3 <the-correct-python-file> --input 9f64b37fe088cea42a3c4a868a278b2d7dd76f2a
And
python3 <the-correct-python-file> --input 9f64b37fe088cea42a3c4a868a278b2d7dd76f2a
Or even without the named arguments:
python3 <the-correct-python-file> 9f64b37fe088cea42a3c4a868a278b2d7dd76f2a

Exercise 4 - C programming

For this exercise, navigate to the C programming folder. In this folder, you will see some *.c files and a Makefile. If you never heard of a Makefile: a Makefile contains and defines compilation "targets". By running for example "make", you can build the application to a binary file. By running "make clean", the application folder gets cleaned up and all files generated during compilation are removed.
The intention of this exercise is to make the program compile. To make it a challenge, we made some compilation errors on purpose in the code for you to solve.

Exercise 5 - Serial ports

In this exercise you will need to run the program that was compiled in exercise 4. Once you run the program, it will print out the name of a serial port. You will need to write the output of exercise 3 to that serial port to start the log of the compiled program (or to stop it, with the second output of exercise 3). The exercise is to get the logs of the program out of the compiled binary by writing the correct data to the serial port manually.



So you need to open a command prompt to send and receive data to and from the compiled C program. In order to run both the c program and a serial terminal, you can open the same docker instance in different terminal windows. You can do this by running the following command in a new terminal:

$ docker exec -it qteal_ctr /bin/bash

Now you have a second terminal session running in the same docker environment.

Exercise 6 - Combining Python and C program

For this exercise you will need to do the same as for exercise 5, but in an automated way. Write a python script that starts up the compiled binary, reads the name of the serial port, writes the start command, reads the binary log, writes the stop command, and stops the binary from running. Print the entire log of the compiled binary to the console of the python program.

Exercise 7 - Parsing logs

Adapt the python program from exercise 6 to parse all timestamps found in the log from the compiled binary. The timestamps in the log are of the format "HH:MM". The exercise is marked as complete if you manage to only print all timestamps to the console.